Quantum Phase Logic in Boolean Formula Satisfiability Problems

Oleksandr Korchenko and Oleh Zaritskyi

National Aviation University, av. L.Guzara, 1, Kyiv, 03124, Ukraine

Abstract

The article deals with the topical issues of quantum phase logic application for solving practical problems of Boolean formulas satisfiability. The use of QPU allows to significantly increase the speed of solving SAT problems as one of the most important research areas that influence the development of such sections of artificial intelligence as brain modeling and cognitive science and their applied areas: formal logic, rules and analogies, problems of satisfiability of logical formulas, theorem proving. The article presents a general algorithm for modeling SAT tasks using QPU and a real-life example of solving the problem of a logical formula satisfiability, discusses topical issues of modeling a quantum system and interpreting the results.

Keywords [1]

Quantum phase logic, quantum amplitude logic, 3-SAT, 2-SAT, propositional logic formulae, conjunctive normal form

1. Introduction

Science and technology have entered the era of new computing platforms built on the fundamental laws of the universe, which will lead to a revolutionary acceleration of solving practical applied problems from the point of view of the computational complexity theory. We are talking about quantum computing, which is part of the field of quantum information science (QIS). The prospects of quantum computers are primarily related to their ability to significantly expand the computing horizons of conventional computing tools, using their own "natural" parallelism of computation in the form of superposition and entanglement of qubits.

Obtaining a quantum superiority is considered for a certain range of tasks, which does not diminish the importance of quantum computers, given that some of these tasks are beyond the computational capabilities of any hypothetical computing device.

The use of quantum computing for solving applied problems is still in its formation and intensive development. According to experts, the most promising areas for the use of quantum computers are chemistry, the study of the properties of new materials, and financial services [1]. Artificial intelligence will also be able to increase the speed of some computing algorithms, for example, in the field of machine learning. Quantum cryptography methods will have been widely developed [2]. The list of applications of quantum computing is constantly growing and expanding due to the research and development of new quantum algorithms and hardware that is becoming available for scientific research.

Among the existing quantum algorithms, we can distinguish the following [3,4]:

- quantum phase estimation;
- complex amplitude amplification;
- quantum Fourier transform (QFT);
- quantum search (QS);
- factorization of integers;
- finding the period of a function;
- eigenvalue estimation;

Information Technology and Implementation (IT&I-2023), November 20-21, 2023, Kyiv, Ukraine
EMAIL: icaocentre@nau.edu.ua (A.1); oleh.zaritskyi@npp.nau.edu.ua (A.2)
ORCID: 0000-0002-6116-4426 (A.1); 0000-0003-3376-0631 (A.2)

- quantum machine learning;
- quantum over-sampling.

The listed algorithms and approaches are not exhaustive, but even this list gives an idea of the practical application of quantum computing, which is usually associated with big data processing, where processing time is critical.

One of the classes of problems that quantum search allows to solve includes problems that must obtain a yes/no answer, that is, derive the value of a traditional logical command and belong to NP-complete problems.

Traditional tasks of this class are searching for a specific value in a database or solving satisfiability problems of Boolean formulas (SAT problems). The search for methods to solve problems of this class is very important in terms of their impact on the development of such areas of artificial intelligence as whole brain emulation (WBE) and cognitive science (CGS). In turn, these areas of AI research and development cover a wide range of tasks: formal logic (FL), rules and analogies (rules), Boolean satisfiability problems (SAT), automated theorem proving (ATP), deep learning (DL) as a basis for natural language processing (NLP) and computer vision (CV), etc. [5].

Due to the exceptional importance and relevance of these subject areas of knowledge for the development of integrated cognitive architectures, the article discusses the peculiarities of the practical implementation of quantum phase logic for solving actual problems of a logical formula satisfiability.

The object of the study is a quantum processor unit (QPU) as an environment for modeling logical algorithms within the framework of the study. The subject of the study is quantum phase logic as a tool for solving the problem of satisfiability of a logical formula (function). The aim of the study is to increase the speed of solving SAT problems by using quantum phase logic implemented in QPU.

2. General formulation of SAT problems, existing approaches

Let us consider a general formulation of satisfiability problems of Boolean formulas in conjunctive normal form (CNF). There is a set X of n Boolean variables that can take one of two values 0 or 1 (or false \ true). A literal on X is one of the variables x_i or its negation $\widetilde{x_i}$ [6]. The condition C_k is the disjunction of literals (1):

$$\begin{gathered} x_i = \{0,1\} \in X, i = [1 \div N] \\ C_k = x_1 \vee x_2 \vee \widetilde{x_3} \vee \ldots \vee x_n, k = [1 \div K] \end{gathered} \quad (1)$$

Logical assignment for X is the assignment of values 0 or 1 to each literal in a condition.

The standard propositional 2,3 satisfiability (2,3-SAT) problem involves formulating a propositional logic formula that consists of a conjunction of literals disjunctions (conditions), where each condition consists of 2,3 literals that make the formula true. That is, the satisfiability problem is to find the values of literals that make the formula true (2):

$$\mathrm{C}_1 \wedge \mathrm{C}_2 \wedge \ldots \wedge C_k = True\ , \quad (2)$$

Satisfiability problems are fundamental problems of combinatorial search, which are among the most difficult computational tasks. It is necessary to find $\boldsymbol{n}$ independent solutions, fulfilling the truth constraints of a Boolean formula. Such problems belong to the class of NP-complete problems and can be solved in polynomial time. As a test example, consider the 3-SAT problem (3):

$$(a \vee b) \wedge (\tilde{a} \vee c) \wedge (b \vee \tilde{c}), \quad (3)$$

In accordance with the statement of the satisfiability problem, it is necessary to find, if they exist, such values of the literals a, b, c that will make the logical formula (3) true.

The implementation of the logic described by formula (3), using logic gates that perform basic logic operations according to the IEC 60617-12:1997 standard, is shown in Fig. 1. There are a number of methods for solving the problems of the satisfiability of logical formulas based on classical deductive methods of proving theorems [6,7]. The DPLL (Davis-Putnam-Logemann-Loveland) algorithm is based on return search and distributed deterministic computing (unit-propagation) [8]. DPLL is a complete, backtracking-based search algorithm for deciding the satisfiability of propositional logic formula in conjunctive normal form, i.e. for solving the CNF-SAT problem.

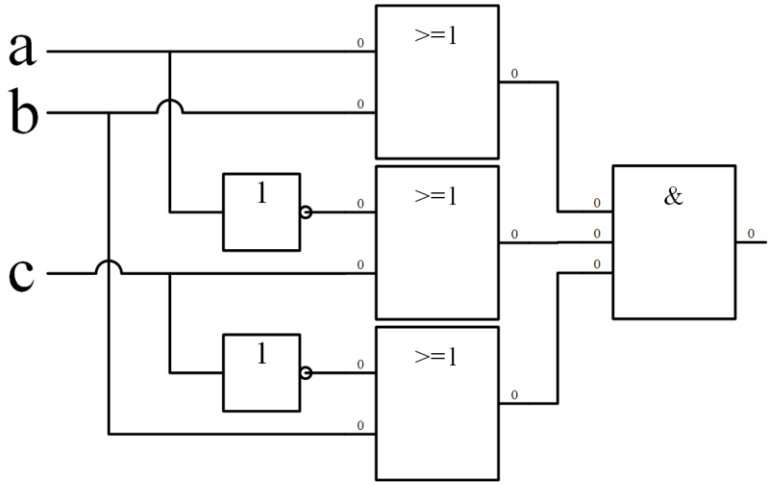

Figure 1: Representation of a SAT problem using basic elements of a digital circuit

The algorithm considers the value of some literal to be true and calculates all the deterministic consequences of this assumption, performing cyclic calculations until it finds a solution. In article [9] is reported that the performance of an enhanced version of the "Davis-Putnam" (DP) proof procedure for propositional satisfiability (SAT) on large instances derived from real world problems in planning, scheduling, and circuit diagnosis and synthesis. The results show that incorporating CSP lookback techniques – especially the relatively new technique of relevance-bounded learning – renders easy many problems, which otherwise are beyond DP's reach. Frequently they make DP, a systematic algorithm, perform as well or better than stochastic SAT algorithms such as GSAT.

GSAT (Greedy SAT) is local, as it makes decisions about the values of literals based on local information only. At the beginning of the algorithm, literals are assigned arbitrary values and the value of the variable is changed if it gives the largest increase in completed sentences.

Methods for solving SAT problems involve their parallelization using CDCL solvers (conflict-driven clause learning) [10,11]. Similar to DPLL, the algorithm makes decisions on literal values and performs deterministic calculations, on the other hand, it keeps the implication graph in memory and remembers some combinations that do not lead to a solution, which increases search efficiency.

Common to the above methods of solving SAT problems is the search by enumerating the values of literals using implication graphs. It is clear that an increase in the number of literals will lead to a quadratic increase in computational complexity, and at a certain number of them, calculations using classical computers will become almost impossible.

3. Quantum phase logic. Algorithm for solving SAT problems using QPU

Let's look at the difference between the different types of logic used in computing systems [12]. Hardware built on the classic von Neumann architecture, which uses bits of information and stores them in short-term and long-term memory, is characterized by the use of traditional binary logic that applies logic gates (Fig.1) to input data to produce a result.

Quantum computing, based on the principles of superposition and qubit entanglement, uses amplitude and phase logic. Quantum amplitude logic applies logic gates to the state of an input register to produce a superposition of results. Quantum phase logic inverts the phase of each qubit of the input register, which yields a 1 as a result of the measurement. In other words, the quantum circuit inverts the relative phases of all input values for which the logic operation is performed.

The construction of quantum algorithms for solving SAT problems is performed using basic digital logic gates built from quantum CNOT gates and QPU operations realizing phase logic and involves a number of consecutive steps [13]:

1. Transformation of the formula from a satisfiability problem to a form consisting of a number of conditions C_k and involving their simultaneous fulfillment, i.e., the conjunction operation (2). This approach reduces the number of service qubits, since the phase conjunction operation can be performed simultaneously for any number of qubits (i.e., literals) using a single CPHASE operation.

2. Represent each condition C_k using amplitude logic. As a rule, one service qubit is created for each condition. As a practical example, consider the preparation of a gate that implements amplitude OR logic (Fig. 2). Such a gate uses two working qubits and one service qubit to obtain the result of an operation. Fig. 2 shows a code fragment of the gate implementation program, its detailed graphical representation and as a subschema ("black box"), as well as the corresponding element of the digital circuit of the Boolean formula (Fig. 1).

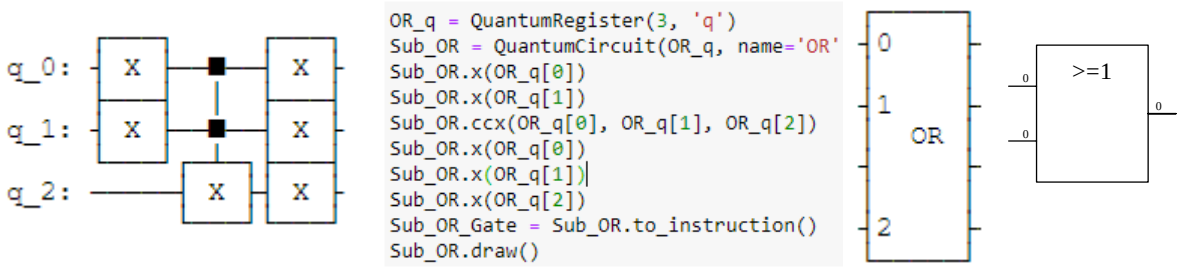

Figure 2: OR gate implemented using quantum operations NOT and CCNOT, code example in Python, Qiskit module (IBM)

On the basis of the gate (Fig. 2), it is possible to implement all three elements of Boolean logic OR (Fig. 1): for the first digital circuit, the inputs $q[0], q[1]$ are supplied with the literals a, b respectively, for the second circuit, the literals, $\tilde{a} \vee c$ for the third circuit, the literals $b \vee \tilde{c}$, respectively. The development of such gates should be done from the point of view of their universality for typical logical operators with the possibility of using them for all the same type of formula conditions. As a rule, such gates are developed as sub circuits and converted into a general scheme for implementing a logical formula, as will be shown in Section 4, using the [subcircuit_name] command.to_instruction()

3. Initiate a full QPU register with the number of qubits equal to the number of literals in a uniform superposition using the J. Adamar valve and initiate all service qubits in the state $|0\rangle$.

4. After realizing all the conditions C_k in the amplitude logic, perform the conjunction operation in the phase logic.

5. Cancel the calculations of all operations in amplitude logic. The operations are canceled in the reverse order from the last to the first condition.

6. Perform a mirror subscheme of the complex amplitude amplification (AA) circuit to select the m states of the input data that ensure making the logical formula to be truth. Due to the need to use the mirror subscheme several times N_{AA} (4) to ensure the amplification of the amplitude of the required combination of n literals in order to clearly detect them, it is also necessary to prepare the subscheme as a typical block (Fig. 3) and convert it into a general scheme for implementing the formula.

$$N_{AA} = \frac{\pi}{4}\sqrt{\frac{2^n}{m}} \tag{4}$$

The implementation of the mirror subscheme of the AA circuit (Fig. 3) involves the use of Adamar, NOT and CCPHASE gates. Phase inverting allows you to select a register value and highlight its phase against the background of others, and the mirror operation converts the phase difference into an amplitude difference.

This combination of operations in quantum computing is called the complex amplitude amplification (AA) iteration.

4. Practical implementation of the 3-SAT problem in the quantum computing environment

The practical implementation of the logical formula (3) was carried out in the Python environment using the Quiskit quantum computing module from IBM [14]. A view of the complete quantum computing scheme is shown in Fig. 4. The quantum circuit consists of [15]:

- quantum input data register, represented by 3 working qubits (a, b, c) - upper register for recording the values of three literals, three service qubits (s[0], s[1], s[2]) - lower register for intermediate calculations;

▪ groups of quantum gates (gates) that perform operations of amplitude and phase quantum logic, cancellation of calculations of operations, mirror subscheme of the amplitude amplification (AA) circuit;

▪ three measurements, the results of which are written to the corresponding classical bits (m1,m2,m3) to save the results of calculations from working qubits.

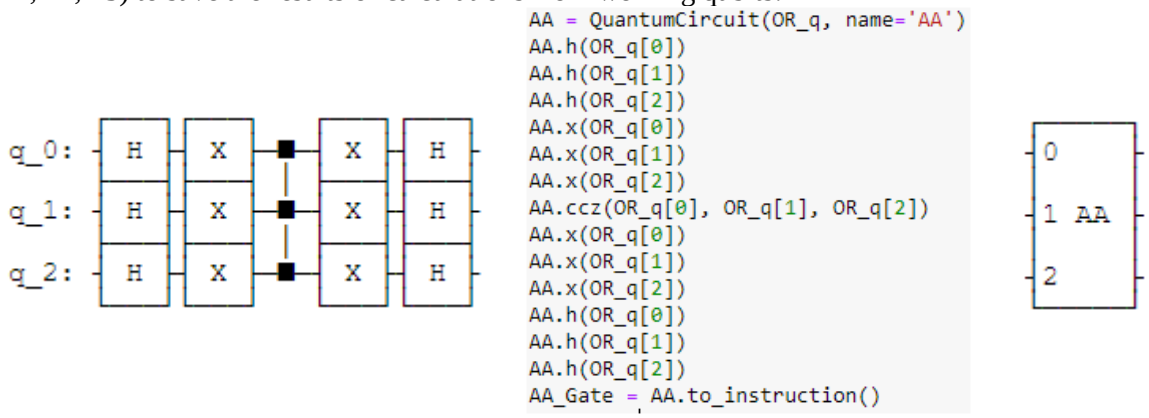

Figure 3: Mirror subcircuit of the amplitude amplification (AA) circuit

The scheme expanded by the [scheme_name].decompose() command gives an idea of the depth and width of the quantum scheme. The depth is 34, i.e., 34 operations are performed from the base state of the register to the moment of measurement. The width of the scheme is 6, which is the number of qubits involved in the calculations, including service qubits. Let's consider several options for modeling the logical formula (3).

Option 1. Modeling is performed without using the mirror subscheme of the amplitude amplification (AA) circuit.

The result of modeling the formula using the 'statevector_simulator' in the register phases is presented both in the standard notation of P. Dirac's state vector of a quantum system and in graphical form on the Bloch sphere (Fig. 5).

After analyzing the results, we can distinguish three states 3, 6, 7: |000010⟩, |000110⟩, |000111⟩, which differ from the others in the relative phase π, as indicated by the «-» sign before their amplitude and the color on the Bloch sphere. That is, the phase has been inverted. The first three digits are not taken into account, they describe the state of the service qubits after canceling the calculations.

These states encode the logical assignment $a = 0, d = 1, c = 0), (a = 0, d = 1, c = 1), (a = 1, d = 1, c = 1)$ respectively, which indicates that the given logical formula can be realized and the obtained sets of literals ensure the satisfiability of the original logical formula. Simple measurement of the results using the 'qasm_simulator' will not solve the SAT problem, since all possible initial states of the system are described by the same amplitude, and the probability of their occurrence does not depend on the sign and is equal to 1/8 (fig.6).

Option 2. Modeling using the mirror subcircuit of the AA circuit.

The result of modeling formula (3) using 'statevector_simulator' in register phases is represented in the standard P.Dirac notation by the state vector of the quantum system (Fig.7). What is obvious, apart from the appearance of the relative phase π, is the increase in the amplitude of the system states encoding the logical assignment after a single application of amplitude amplification.

It should also be noted that the AA circuit inverted the phase of the qubits before amplifying the amplitude and set it to 0, leaving the relative phase π unchanged. Since the AA circuit in this case amplified the amplitude, the simulation results using the 'qasm_simulator' already allow us to make conclusions about the states in which the logical assignment is encoded with a certain probability ($(-0.53033)^2 \approx 0.282$; $2825 \div 10000 \approx 0.283$) (Fig. 8). Using formula (4), we obtain the number of iterations of using the mirror subscheme of the AA circuit:

$$N_{AA} = \frac{\pi}{4}\sqrt{\frac{2^3}{3}} \approx 1{,}28$$

Thus, for formula (3), one iteration of the mirror subscheme of the AA circuit is sufficient to obtain a result that will allow us to unambiguously identify the states of the quantum system in which the logical assignment for formula (3) is encoded.

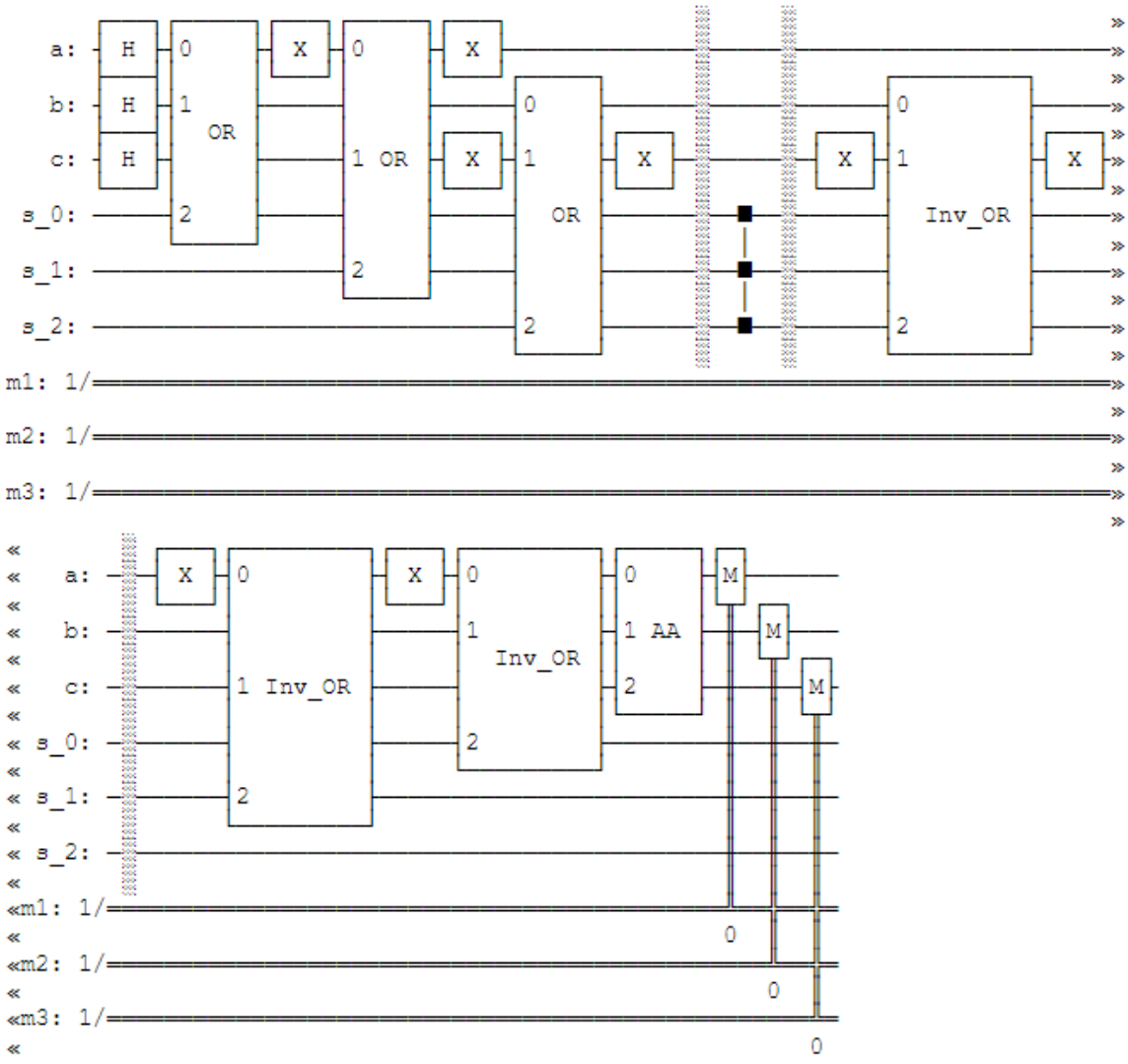

Figure 4: Quantum scheme for calculation of the logical formula (3)

```
from qiskit import *
simulator = Aer.get_backend('statevector_simulator')
result = execute(SAT, backend = simulator).result()
statevector = result.get_statevector()
statevector.draw('latex')
```

$$\frac{\sqrt{2}}{4}|000000\rangle + \frac{\sqrt{2}}{4}|000001\rangle - \frac{\sqrt{2}}{4}|000010\rangle + \frac{\sqrt{2}}{4}|000011\rangle + \frac{\sqrt{2}}{4}|000100\rangle + \frac{\sqrt{2}}{4}|000101\rangle - \frac{\sqrt{2}}{4}|000110\rangle - \frac{\sqrt{2}}{4}|000111\rangle$$

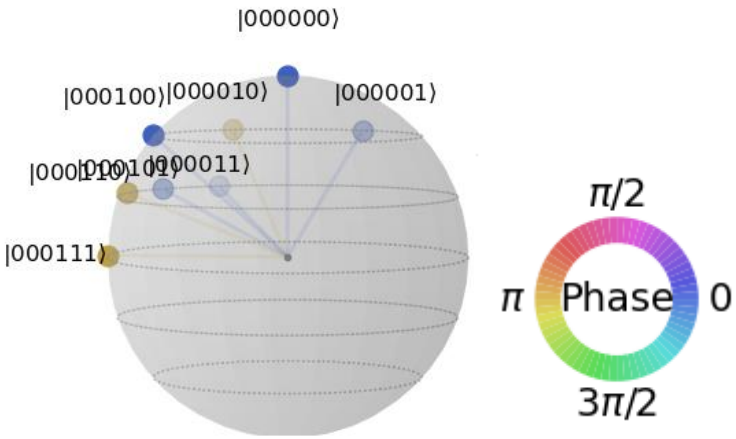

Figure 5: Simulation results without AA scheme

```
simulator = Aer.get_backend('qasm_simulator')
result = execute(SAT, backend = simulator, shots =10000 ).result()
plot_histogram(result.get_counts())
```

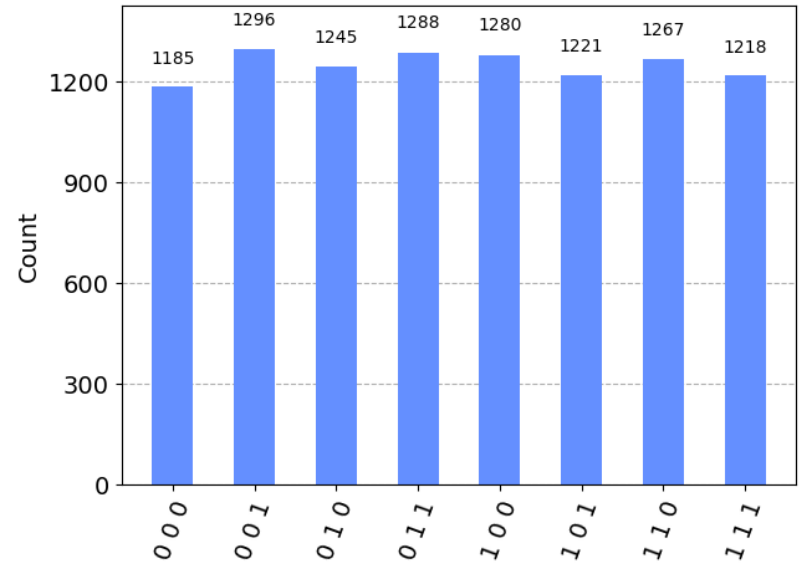

Figure 6: Probability of system states after modeling

```
from qiskit import *
simulator = Aer.get_backend('statevector_simulator')
result = execute(SAT, backend = simulator).result()
statevector = result.get_statevector()
statevector.draw('latex')
```

$$0.1767766953|000000\rangle + 0.1767766953|000001\rangle - 0.5303300859|000010\rangle + 0.1767766953|000011\rangle + 0.1767766953|000100\rangle + 0.1767766953|000101\rangle - 0.5303300859|000110\rangle - 0.5303300859|000111\rangle$$

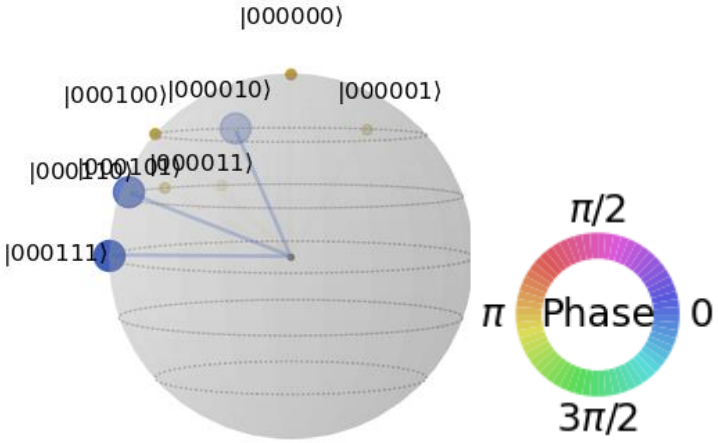

Figure 7: Modeling results with a single application of the AA scheme

5. Summary and Conclusion

As shown, simulations using the mirror subcircuit of the amplitude amplification circuit allow us to identify with high probability (Fig.8), after qubit measurements, the logic assignments encoded in the system states that make the logic formula true. It is also possible to identify the necessary encoded states without performing measurements by analyzing the relative phase, which is equal to π (Fig.5). The proposed simplified algorithm for constructing a logic function model with QPU using standardized sets of gates to implement amplitude and phase logic allows to implement a logic formula of any complexity. Since the quantum phase logic operation for the conjunctive normal form can be performed simultaneously for any number of qubits (i.e., literals) using a single CPHASE operation, the computation speed will significantly exceed existing algorithms. The search for the solution of SAT problems with a certain probability, for traditional algorithms is carried out in time $O\left(k^n poly(n)\right)$,

because it is necessary to implement the procedure of searching for the values of literals k times for n literals. To speed up the traditional algorithms it is necessary to fulfill them with the help of quantum computing, replacing the probabilistic search procedure with the complex amplitude amplification algorithm, which will allow to solve the problem in $O(1.1(53^n poly(n)))$ time.

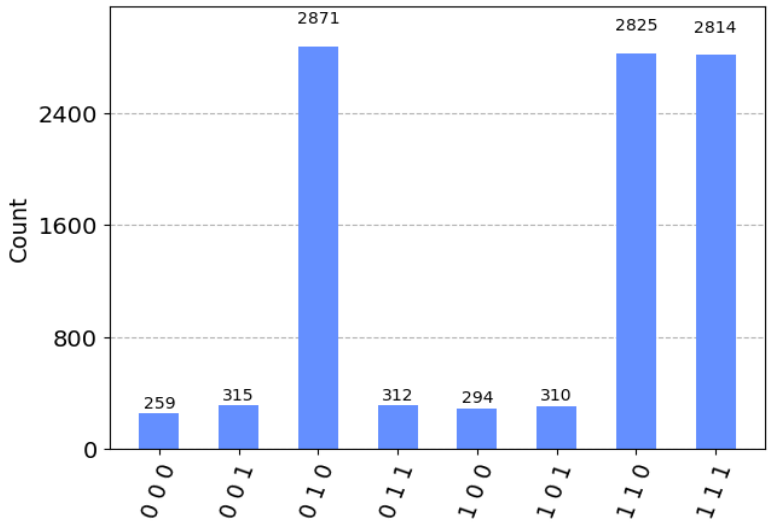

Figure 8: Probability of system states after modeling

6. References

[1] J.D. Hidary. Quantum computing: An applied approach. Springer. – 2019. – 370 p.
[2] V. Silva. Practical Quantum computing for developers. Apress. – 2020. – 352 p.
[3] P.Kaye, R.Laflamme, M.Mosca. In introduction to Quantum computing. Oxford University Press. – 2007. – 284 p.
[4] R.S. Sutor. Dancing with qubits. How quantu, computing works and how it can change the world. Packt Publishing. – 2019. – 512 p.
[5] O.Zaritskyi, O.Ponomarenko. Quantitative assessment of technological singularity. The International Scientific and Technical Journal Problems of control and informatics, 2022. - №1. - P.93 – 111. DOI: http://doi.org/10.34229/1028-0979-2022-1-9.
[6] J. Lkeinberg, Eva Tardos. Algorithm design.Cornell University. – 800 p.
[7] Biere, Heule, Van Maaren, Walsh (February 2009). Handbook of Satisfiability. IOS Press. p. 138.
[8] Nieuwenhuis, Robert; Oliveras, Albert; Tinelli, Cesar (2004), "Abstract DPLL and Abstract DPLL Modulo Theories", Proceedings Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 2004, pp. 36 – 50.
[9] Roberto J. Bayardo Jr.; Robert C. Schrag (1997). "Using CSP look-back techniques to solve real world SAT instances". Proc. 14th Nat. Conf. on Artificial Intelligence (AAAI). pp. 203–208.
[10] Marques-Silva J. P. GRASP: A search algorithm for propositional satisfiability / J.P. Marques-Silva, K. A. Sakallah // IEEE Transactions on Computers. – 1999. – Vol. 48, N 5. – P. 506 – 521.
[11] J.P. Marques-Silva; Karem A. Sakallah (May 1999). "GRASP: A Search Algorithm for Propositional Satisfiability". IEEE Transactions on Computers. 48 (5): 506–521. doi:10.1109/12.769433.
[12] T.G. Wong. Introduction to classical and quantum computing. Rooted Grove, Omaha, Nebraska. – 2022. – 400 p.
[13] E.R.Jonsston, N.Harrigan, M.Gimeno-Segova. Programming Quantum computers. Essential algorithms and code samples. O'Reilly. – 2021. – 336 p.
[14] S.Kaiser, C.Granade. Learn quantum computing with Python and Q#. A hands-on approach. Manning Publication. – 2012. – 430 p.
[15] M. Nielsen, I.Chuang. Quantum computation and quantum information. Cambridge University Press. – 2011. – 824 p.